Algorithms 2

Harish Manoharan, Ishan Pendse
Palak Singh, Shone Pansambal

Problem Statement

Given a set of text documents (without any internal hierarchy), find out the pairs (or groups) of documents that are significantly similar in the sense of possible plagiarism.

Our Approach

- 23rd May- 5th June: Reading the provided material and other information.
- 5th June- 10th June: Deciding language to use, literature reading, finding a dataset.
- 11th June-20th June: Completed our first pilot model.

First Pilot Model:

- We used a small dataset of 12 assignments of a BS course.
- It had 3 information based questions and was an open book/ open Internet assignment
- We used NLTK for preprocessing, and the similarity measures of Cosine Similarity and Levenshtein Distance for comparison.
- We did separate analysis for each question in the document.
- We compared all documents pairwise and assigned a score to each pair. This was used to determine suspicious pairs which were manually checked to gauge the effectiveness of our methods.

Edit Distance

- For the the prototype using levenshtein distance,we initially separated each assignment into three documents having the answers for question 1,2 and 3 which represent structure, entry and treatment of coronavirus.
- In the preprocessing step, every document was converted into a list having each sentence in it as the elements. Also all the sentences were processed by removing the spaces and punctuation marks and thus converted into strings.
- Then we applied the levenshtein distance function to each pair of sentences in every pair of document for each question. All sentence pairs with Levdistance/No. Of Characters ratio less than a threshold value were considered, and the average value of this ratio for these sentences was assigned as a score to the particular pair of documents.
- While comparing two sentences with m and n characters each, our Levenshtein program function has O(mn) time complexity. We used a bottom-up dynamic programming approach. We constructed the matrix row-wise to reduce the space complexity to O(min(m,n))
- Optimizations employed:
  a. Condensing: We clubbed a specific number ‘d’ characters together and considered it as a single character while calculating Levenshtein distance. This reduces the time complexity to O(mn/d^2). We are now thinking about using words as a single unit in computing Levenshtein distance between two sentences which will improve running time even more and also is logically more correct to do.
  b. Stopped the execution of Levenshtein distance as soon the value crossed the threshold, which improved runtime.

Results

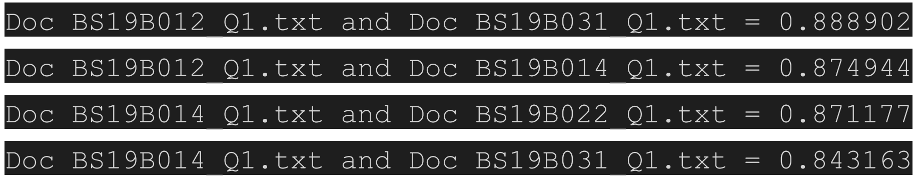
- Tested 4 highest ranked document pairs from each question, thus total 12 document pairs.
- Out of these,
  - 2 Pairs: Very High similarity, Copy Paste type
  - 6 Pairs: Moderate similarity, Copy Paste type
  - 2 Pairs: Low Similarity, Copy Paste type
  - 2 Pairs: No Similarity
- This shows that Levenshtein distance is decently successful in detecting copy and paste/ near copy paste plagiarism.
- We also think that in addition to this, it will also be useful to detect summarized plagiarism.
- This method will fail in highly obfuscated plagiarism, though in some cases it can be salvaged by doing appropriate preprocessing.

Cosine Similarity

- Convert each text document into a word frequency vector.For ex:
  “the dog barked at the man”={‘the’-2,’dog’-1,’barked’-1,’at’-1,’man’-1}
  “Man pet the dog”={‘man’-1,’pet’-1,’the’-1,’dog’-1}
- Take all possible pairs of documents and find the dot product of their corresponding vectors
  While taking dot product, each word is considered as a different axis.
  Cosine similarity=cosθ=DotProduct(A,B)/(|A|*|B|)
  Cosine similarity=(2*1+1*1+1*1)/(sqrt(8*4)=0.707
  Dot product ranges from 0 to 1. More the dot product more the similarity.
- The dot products are stored in a 2-D array where ans[i][j]=similarity measure between doc i and doc j.

Results

Since we have not implemented a clustering algorithm yet, top four pair of docs(by cosine similarity measures) were taken...each pair was manually checked for plagiarism and the results were as follows



```
Doc BS19B012_Q1.txt and Doc BS19B031_Q1.txt = 0.888902
Doc BS19B012_Q1.txt and Doc BS19B014_Q1.txt = 0.874944
Doc BS19B014_Q1.txt and Doc BS19B022_Q1.txt = 0.871177
Doc BS19B014_Q1.txt and Doc BS19B031_Q1.txt = 0.843163
```

Out of these four only the first one was slightly plagiarised...The results for the other two questions were quite similar...The reasons are as follows:

- Some of them had a few lines copy pasted...since the rest was different and some of the docs had more words, cosine similarity was unable to detect them but levenshtein distance did.
- The dataset was small and the docs had less words..It would work better with larger docs
- The questions were specific and most of the answers had the same biological terms which resulted in high similarity values...this will be improved with tf-idf vectors

Further Course of Action

1. We are planning to test the program on a larger number of documents, and need a robust dataset which has a fair chance of plagiarism during its creation. We also would like to find a dataset which also provides all the instances of plagiarism. In this case we can compare our results with theirs and calculate an accuracy score.
2. Synonyms handling: A part of preprocessing which will replace all the synonyms in the text document with a common word. This will improve the score of any comparison method used.
For this, we can use the “WordNet” database.
3. Using Words as a unit in Levenshtein distance, thinking about giving weights to different edit steps.
4. Improving Preprocessing: We will work on improving the preprocessing of the program, which includes stop word removal, lemmatization, stemming.
Also, we will work on reducing the time taken to run the program, as the time complexity of the current Levenshtein algorithm is such that the time taken would become infeasible on increasing the number of text documents.

Further Course of Action

5. Better way than pairwise approach?: In both of our current methods, we have assigned a similarity score to each pair of documents. We will try to figure out a way for dealing with multiple documents at the same time. This may lead to improvements in execution time.
6. Clustering: After comparing the text documents, we plan to apply a clustering algorithm to divide the set of documents into groups of similar documents. An interesting point can be to try and figure out the directionality of copying.
7. Different methods for comparison: We are searching for more other efficient method for plagiarism is detection like hashing/substring matching which may take less time and provide more accurate results.
8. Narrowing down the Problem Statement: We observed that the method of plagiarism detection to be adopted highly depends upon the type of plagiarism present. The types of plagiarism, in turn depend upon the environment in which the documents were created.